#17 / Appeal Brief
T. McBeth 9/17/03





**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**
**BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

| | | | |
|---|---|---|---|
| In re Application of: | Harlan SEXTON et al. | | |
| Application No.: | 09/248,294 | Group Art Unit: | 2126 |
| Filed: | February 11, 1999 | Examiner: | Zhen, L. |
| Attorney Docket: | 50277-0179 | | |
| Client Docket: | OID-1997-048-03 | | |

For: ADDRESS CALCULATION OF INVARIANT REFERENCES WITHIN A RUN-TIME ENVIRONMENT



**APPEAL BRIEF**

Honorable Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

This Appeal Brief is submitted, in triplicate, in support of the Notice of Appeal filed July 8, 2003.

## I. REAL PARTY IN INTEREST

Oracle International Corp. is the real party in interest.

## II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals and interferences.

## III. STATUS OF THE CLAIMS

Claims 1-26 are pending in this appeal. Claims 4, 10, 14, 20-21, and 24 are indicated as allowable. This appeal is therefore taken from the final rejection of claims 1-3, 5-9, 11-13, 15-19, 22-23, and 25-26 on April 7, 2003.

## IV. STATUS OF AMENDMENTS

The amendment to claim 25 filed May 23, 2003 has not been entered and is not relied upon in this appeal. However, another amendment to claim 25 is filed herewith to address an objection.

## V. SUMMARY OF THE INVENTION

The present invention addresses problems associated with implementing a run-time environment, for example, a virtual machine for a language such as JAVA. Typically, run-time environments implement a program state with machine-specific references such as machine pointers, but machines are not portable to different processes, for example to store the program state on disk or another secondary storage medium and restore the stored program state to main memory. (Spec., p. 3) Accordingly, the present invention addresses need for an invariant reference storage format that is machine independent or capable of being transferred quickly between different processes, but also suitable for use as a run-time storage format. (Spec., p. 5)

These and other needs are addressed by the present invention by providing a self-relative numeric reference format for run-time storage of references. The representation of a numeric reference is invariant because it is an integer value and can be easily and efficiently converted into a machines pointers and back. Furthermore, a self-relative numeric reference, which encodes the location of an object relative to a pointer to a referencing object that contains the self-relative

numeric reference, is efficient because the pointer to the referencing object is readily available. (Spec., p. 7)

This is reflected in claim 1, with reference to the figures, as follows: A method of generating a first tagged machine pointer **P** to a first object **216** referenced by a second object, **214** said method comprising the computer-implemented steps of: fetching **500** a tagged numeric reference **300** stored within the second object **214** based on **502** a second tagged machine pointer **S** that points to the second object **214**; and generating **504** the first tagged machine pointer **P** as a sum including the tagged numeric reference **300** and the second tagged machine pointer **S**.

This is also reflected in claim 5, with reference to the figures, as follows: A method of managing memory, comprising the computer-implemented steps of: storing a first object **214** and a second object **216** in a memory **210**, wherein the first object **214** and the second object **216** do not overlap each other; and storing **404** a reference **300** within a first object **214** to a second object **216** in the memory **210** as a numeric reference **300** that encodes a location of the second object **216** as an offset **P-S** from an address of the first object **214** in the memory **210**.

## VI. ISSUES

Whether claims 5, 15, 23, and 26 are anticipated under 35 U.S.C. § 102 by *Lee* (US 6,345,276)?

Whether claims 1-2, 6-8, 11-12,16-18, 22 and 25 are obvious under 35 U.S.C. § 103 based on *Lee* (US 6,345,276) in view of *Murray* (Kelly E. *Murray*, "Under the Hood: CLOS")?

Whether claims 3, 9, 13, and 19 are obvious under 35 U.S.C. § 103 over *Lee* and *Murray* further in view of *Carter et al.* (US 6,003,123).

## VII. GROUPING OF CLAIMS

The claims should not be regarded as all standing together since the claims recite respective limitations that render each of the claims separately patentable. For the purposes of this appeal, the following groups are recognized:

A. Claims 1-2, 6-8, 11-12,16-18, 22 and 25

B. Claims 3, 9, 13, and 19

C. Claims 5, 15

D. Claims 23, and 26

## VIII. ARGUMENT

### A. CLAIMS 5, 15, 23, AND 26 ARE NOT ANTICIPATED BY *LEE*, BECAUSE *LEE* FAILS TO DISCLOSE "STORING A REFERENCE ... AS A NUMERIC REFERENCE THAT ENCODES A LOCATION OF THE SECOND OBJECT AS AN OFFSET FROM AN ADDRESS OF THE FIRST OBJECT IN THE MEMORY."

To anticipate a patent claim, every element and limitation of the claimed invention must be found in a single prior art reference, arranged as in the claim. *Karsten Mfg. Corp. v. Cleveland Golf Co.*, 242 F.3d 1376, 1383, 58 USPQ2d 1286, 1291 (Fed. Cir. 2001); *Scripps Clinic & Research Foundation v. Genentech, Inc.*, 927 F.2d 1565, 1576, 18 USPQ2d 1001, 1010 (Fed. Cir. 1991).

The rejection of claims 5, 15, 23, and 26 must be reversed, because *Lee.* does not disclose the limitations of the claims.

For example, independent claims 5 and 15 recite:

> **storing** a reference within a first object to a second object in the memory as a numeric reference that **encodes a location of the second object** as an **offset** from an address of the first object in the memory

This limitation is not found in *Lee*. Rather, *Lee* discloses a smart pointer that "contains two based addresses which are offsets **relative to the start of a shared memory heap**" (Abstract, emphasis added), not the "address of the first object in the memory" as recited in independent claims 5 and 15.

Specifically, with reference to FIG. 2, object **203** at offset 20 in the heap **201** contains a smart pointer **204** to object **207** at offset 300 in the heap **201**. The Examiner correctly calculates that the "the difference of fields 205 and 206 (300-20=**280**) is the **offset** of object1 207 from the address of object0 203." (Page 7, lines 7-9, emphasis added) However, this offset value of 280 is plainly not stored anywhere in FIG. 8 as a reference as required by this limitation or anywhere else. Smart pointer **204** actually stores the value 20 in portion **205** (the offset of the offspring pointer **204** from the start of the heap **201**), and the value 300 in a destination-pointer portion **206** (the offset of the object **207** in the heap **201**). Even though the difference can be computed from these fields of the smart pointer, that difference is not stored in the smart pointer. Therefore, *Lee* does not disclose "**storing** a reference within a first object to a second object in the memory as a numeric reference that **encodes a location of the second object** as an **offset** from an address of the first object in the memory."

The Advisory Action of June 9, 2003, takes issue with the fact that the offset is not stored anywhere in the disclosure of *Lee* by arguing that "Lee stores the representation of offset (difference of fields 205 and 206 in offspring pointer 204, Fig. 2) in fields 205 and 206, of the offspring pointer 204, Fig. 2." However, neither portion **205** nor portion **206** is an "offset from an address of the first object in memory," and taken together they are merely two different offsets from the start of the heap sitting right next to each other, as stated on col. 6:6-11 (with added emphasis):

> As further shown in FIG. 2, the offspring pointer 204 is a data structure comprising two fields 205, 206. The field 205 ("this-pointer") contains the relative **offset** of the offspring pointer 204 **from the start of the heap 201** and the field 206 ("destination-pointer") contains the relative offset, 300, of the object1 207 **from the start of the heap 201**.

Furthermore, the only concept of an offset disclosed in *Lee* is from the start of the heap, and this offset, i.e., the difference from the start of the heap, is represented as a single word. There is no disclosure in *Lee* of an offset defined from the start of one object to another at all, much more such an offset would need to be represented. Even so, the only representation of offsets disclosed in *Lee* is a single word. There is no representation of an offset acknowledged as such in *Lee* that comprises two separate words. Accordingly, there is no factual basis in *Lee* for the Examiner's assertion that the conjunction of portion **205** and portion **206** is really meant to be a stored representation the offset as defined in claims 5, 15, 23, and 26.

Finally, the rejection cannot be salvaged by reading the recited "first object," not on the object **203**, but on the entire heap **201**, because claims 5 and 15 further recite that "the first object and the second object do not overlap each other but the heap of *Lee*, however, **201** overlap both object **203** and object **207**.

### B. CLAIMS 23 AND 26 ARE ALSO NOT ANTICIPATED BY *LEE*, BECAUSE *LEE* FAILS TO DISCLOSE THAT THE FIRST AND SECOND OBJECTS ARE STORED ON DIFFERENT PAGES.

Claims 23 and 26, which depend from claims 5 and 15, respectively, further recite that the "the memory is subdivided into a plurality of pages; the first object is stored on a first page; and the second object is stored on a second page, other than the first page." Pursuant to 35 U.S.C. § 112, ¶ 4, by virtue of their dependency, claims 23 and 26 also comprise "storing a reference within a first object to a second object" as recited in claims 5 and 15, respectively. Thus, stored within the first object on one page is a reference to a second on another page.

This feature is not detailed by *Lee*. In fact, *Lee* fails to disclose any memory "pages" at all, and the smart pointers only work within the same heap.

The Examiner's short rejection is inadequate with respect to the claim language: "As to claims 23 and 26, Lee (column 5, line 48 – column 6, line 5) [*sic*, shows?] a plurality of pages (virtual memory address space) and storing objects on the pages (see Fig. 2)." Claims 23 and 26 do not merely recite "storing objects on the page," but a specific relation between the first object and the second object, in which the first object has stored within a reference to the second object. The specific relationship is: "the first object is stored on a first page; and the second object is stored on a second page, other than the first page."

All the objects in FIG. 2 of *Lee* that have references therebetween are in the same heap, and *Lee* does not show any "smart pointer" linking objects in different heaps. This is not surprising, since the address calculation offered by *Lee* does not work if the objects are in different heaps. In fact, permitting the referencing and referenced object of a smart pointer to be in different heaps would contradict the disclosure of *Lee*: "Both addresses are based on the start address of the shared memory heap that contains the objects." (col. 10:31-33). *Lee*'s system is premised on the situation that both objects referenced by a smart pointer are in the same heap.

### C. CLAIMS 1-3, 6-9, 11-13, 16-19, 22, AND 25 ARE NOT RENDERED OBVIOUS BY *LEE* AND *MURRAY* BECAUSE THE COMBINATION WOUND NOT RESULT IN EITHER THE CLAIMED INVENTION OR AN OPERABLE EMBODIMENT.

The initial burden of establishing a *prima facie* basis to deny patentability to a claimed invention under any statutory provision always rests upon the Examiner. *In re Mayne*, 41 USPQ2d 1451 (Fed .Cir. 1997); *In re Deuel*, 34 USPQ2d 1210 (Fed. Cir. 1995); *In re Bell*, 26 USPQ2d 1529 (Fed. Cir. 1993); *In re Oetiker*, 24 USPQ2d 1443 (Fed. Cir. 1992). In rejecting a

claim under 35 U.S.C. § 103, the Examiner is required to provide a factual basis to support the obviousness conclusion. *In re Warner*, 154 USPQ 173 (CCPA 1967); *In re Lunsford*, 148 USPQ 721 (CCPA 1966); *In re Freed*, 165 USPQ 570 (CCPA 1970). The Examiner is required to show that all the claim limitations are taught or suggested by the references. *In re Royka*, 180 USPQ 580 (CCPA 1974); *In re Wilson*, 165 USPQ 494 (CCPA 1970).

Obviousness rejections require some evidence in the prior art of a teaching, motivation, or suggestion to combine and modify the prior art references. See, e.g., *McGinley v. Franklin Sports, Inc.*, 262 F.3d 1339, 1351-52, 60 USPQ2d 1001, 1008 (Fed. Cir. 2001); *Brown & Williamson Tobacco Corp. v. Philip Morris Inc.*, 229 F.3d 1120, 1124-25, 56 USPQ2d 1456, 1459 (Fed. Cir. 2000); *In re Dembiczak*, 175 F.3d 994, 999, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999). If a proposed modification would render the prior art being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984).

With respect to the rejections of claims 1-3, 6-9, 11-13, 16-19, 22, and 25 under § 103, there is no motivation to combine *Lee* and *Murray* to produce an invention having the following element: "generating the first tagged machine pointer as a **sum** including the tagged numeric reference and the second tagged machine pointer." Specifically, modifying *Lee* to use the pointer tags of *Murray* would not result either in the claimed invention or in an operable embodiment. *Murray* discloses a Lisp implementation in which the bottom two bits of a machine word indicate the type of the object (p. 82, col. 3). Page 82, col. 1, show an assignment of tag values, in which 00 is for an integer, 01 is for a list point, 10 is for an other pointer, and 11 is nil.

Use of *Murray*'s tag values does not work in the *Lee* system, because the address calculation of *Lee* if combined with *Murray* would produce a pointer to a destination object that

are tagged with the type of the source object. In other words, the pointer to would be tagged with the type of the **wrong object**. For example, in *Lee*, the address of object **207** is calculated in two steps, **317** and **319**. In step **317**, a base pointer is calculated by subtracting offset **205** from pointer **203**. Since offset **205** is a number, its tag according to *Murray* would be 00, and the base pointer would therefore have the same tag as pointer **203**. In step **319**, the base pointer is added to the destination offset **206** to produce the what should be, but is not, the tagged pointer to object **207**. Since the destination offset **206** is also a number, its tag is 00 and the result of step **319** is a pointer with the same tag as the base pointer, which has the same tag as the pointer to object **203**. However, object **207** can have a different type from that of object **203**, so the object **203**'s tag is wrong with respect to object **207**'s type. In fact, there is no assignment of type tags in *Murray* that would result in the generated pointer to object **207** always having the right type tag. Accordingly, combining *Lee* with *Murray* results in an inoperable system in which all destination pointers are tagged with the type of the source pointers.

It is possible to fix the tag destination pointer but only by deviating from the claim limitations. For example, the source type tag may masked off and the correct destination type tag may logically or'd in. But this result is not a "sum." As a result, a functioning combination of *Lee* and *Murray* would not have generated a "first tagged machine pointer as a **sum** including the tagged numeric reference and the second tagged machine pointer." Different operations would have to be employed and those operations destroy the fact that the first tagged machine pointer is a "sum." The claim language "generating ... as a sum" is what, contrary to the Advisory Action, precludes the use of operations, such as masking, that do not generate a sum.

The Examiner's response in the Office Action of April 7, 2003, that "*Lee* does not appear to require objects to be of different types," (p. 7, last line), if true, invalidates the Office Action's

stated motivation on pp. 4-5 to combine *Lee* with *Murray* in the first place: "It would have been obvious the pointers as taught by *Lee* would also be tagged because the tag values of a pointer can provide important information such as distinguishing between pointers and non-pointers. (Column 3, first paragraph, p. 82 of *Murray*.)" The cited passage of *Murray* states: "Type information plays a central role in Lisp implementation. It is used by the garbage collector to distinguish between pointers and non-pointers, and is also required to support heterogeneous collections and polymorphism." (Page 82, column 3) The only basis for the Examiner's combination of *Lee* and *Murray* is the type information, but adding pointers tagged with types as in *Murray* does not work with *Lee*'s pointer arithmetic.

**D. CLAIMS 3, 9, 13, AND 19 ARE NOT RENDERED OBVIOUS BY *LEE*, *MURRAY*, AND *CARTER ET AL.* BECAUSE NONE OF THESE REFERENCES TEACH A TAG PORTION THAT "INDICATES WHETHER THE FIRST OBJECT HAS A SAME OR A DIFFERENT CONTINGUITY AS A CONTIGUITY OF THE SECOND OBJECT.**

Neither *Lee* nor *Murrary* show a tag portion that "indicates whether the first object has a same or different contiguity as a contiguity of the second objection" as recited in claims 3, 9, 13, and 19. *Carter et al.* too fails to teach this feature.

At best, *Carter et al.* shows two different virtual page identifiers, in two different Global Translation Lookaside Buffer (GTLB) entries, which only indicate the number of respective virtual pages (cols. 17:62-67 and 18:15-23). Merely knowing the virtual page numbers is not enough to indicate whether the first or second object is contiguous or discontiguous or whether the contiguity they have is the same or different.

The meaning eventually proposed by the Examiner in the Advisory Action of June 7, 2003, contiguity as "the state of being next or near in time and sequence" does not read upon the *Carter et al.* virtual page number. A virtual page number is not a state of being next or near in

time and sequence; it is just the number of a virtual page. Even if "contiguity" can be read upon a virtual page number, which is highly dubious, *Carter et al.* shows only various different virtual page numbers, but no value that indicates specifically whether the virtual page numbers are the same or different. The claim language requires a specific indication of being the same or different.

The flaw in the Examiner's reasoning can be readily seen from the rejection: "It would have been obvious to apply the teaching of including **contiguity information** in a tag portion of a pointer as taught by Carter." (p. 6, emphasis added). Here, the Examiner generalizes both the claim language's "a same or a different contiguity" and the virtual page numbers of *Carter et al.* under the nebulous rubric of "contiguity information," finds a correspondence, and makes the rejection. This is improper, because claims 3, 9, 13, and 19 do not recite "contiguity information", but rather a tag portion that "indicates whether the first object has a same or different contiguity as a contiguity of the second objection." Even if the *Carter et al.* virtual page number can be twist under the rubric of "contiguity information," that is not what claims 3, 9, 13, and 19 specifically recite. What must be compared with *Carter et al.* is the claim language.

## IX. CONCLUSION AND PRAYER FOR RELIEF

Appellants, therefore, request the Honorable Board to reverse each of the Examiner's rejections.

Respectfully Submitted,

DITTHAVONG & CARLSON, P.C.

9/5/2003
Date

Stephen C. Carlson
Attorney for Applicant(s)
Reg. No. 39929

10507 Braddock Rd, Suite A
Fairfax, VA 22032
Tel. 703-425-8516
Fax. 703-425-8518

## APPENDIX

1. A method of generating a first tagged machine pointer to a first object referenced by a second object, said method comprising the computer-implemented steps of:

fetching a tagged numeric reference stored within the second object based on a second tagged machine pointer that points to the second object; and

generating the first tagged machine pointer as a sum including the tagged numeric reference and the second tagged machine pointer.

2. The method of claim 1, wherein the sum further includes a predetermined constant.

3. The method of claim 1, wherein the tagged numeric reference includes a tag portion that indicates whether the first object has a same or a different contiguity as a contiguity of the second object.

4. The method of claim 3, wherein:

the tag portion includes N bits of the first tagged numeric reference that are less significant than bits used for an offset portion; and

the tag portion contains one of at least a first tag value indicating that the first object is contiguous and a second tag value indicating that the second object is non-contiguous, wherein a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^{N}$.

5. A method of managing memory, comprising the computer-implemented steps of:

storing a first object and a second object in a memory, wherein the first object and the second object do not overlap each other; and

storing a reference within a first object to a second object in the memory as a numeric reference that encodes a location of the second object as an offset from an address of the first object in the memory.

6. The method of claim 5, further comprising the step of calculating a pointer difference between a first machine pointer to the first object and a second machine pointer to the second object to produce the numeric reference.

7. The method of claim 5, wherein the first machine pointer is a first tagged machine pointer, the second machine pointer is a second tagged machine pointer, and the numeric reference is a tagged numeric reference.

8. The method of claim 7, wherein the pointer difference further includes a predetermined constant.

9. The method of claim 7, wherein a tag portion of the tagged numeric reference indicates whether the first object has a same or a different contiguity as a contiguity of the second object.

10. The method of claim 9, wherein:

the tag portion includes N bits of the tagged numeric reference that are less significant than bits used for an offset portion of the tagged numeric reference; and

the tag portion contains one of at least a first tag value indicating that the first object is contiguous and a second tag value indicating that the second object is non-contiguous, wherein a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^N$.

11. A computer-readable medium bearing instructions for generating a first tagged machine pointer to a first object referenced by a second object, said instructions arranged, when executed, to cause one or more processors to perform the steps of:

fetching a tagged numeric reference stored within the second object based on a second tagged machine pointer that points to the second object; and

generating the first tagged machine pointer as a sum including the tagged numeric reference and the second tagged machine pointer.

12. The computer-readable medium of claim 11, wherein the sum further includes a predetermined constant.

13. The computer-readable medium of claim 11, wherein the tagged numeric reference includes a tag portion that indicates whether the first object has a same or a different contiguity as a contiguity of the second object.

14. The computer-readable medium of claim 13, wherein:

the tag portion includes N bits of the first tagged numeric reference that are less significant than bits used for an offset portion; and

the tag portion contains one of at least a first tag value indicating that the first object is contiguous and a second tag value indicating that the second object is non-contiguous, wherein a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^N$.

15. A computer-readable medium bearing instructions for managing memory, said instructions arranged, when executed, to cause one or more processors to perform the steps of:

storing a first object and a second object in a memory, wherein the first object and the second object do not overlap each other; and

storing a reference within a first object to a second object in the memory as a numeric reference that encodes a location of the second object as an offset from an address of the first object in the memory.

16. The computer-readable medium of claim 15, said instructions further arranged to cause said one or more processors to perform the step of calculating a pointer difference between a first machine pointer to the first object and a second machine pointer to the second object to produce the numeric reference.

17. The computer-readable medium of claim 15, the first machine pointer is a first tagged machine pointer, the second machine pointer is a second tagged machine pointer, and the numeric reference is a tagged numeric reference.

18. The computer-readable medium of claim 17, wherein the pointer difference further includes a predetermined constant.

19. The computer-readable medium of claim 17, wherein a tag portion of the tagged numeric reference indicates whether the first object has a same or a different contiguity as a contiguity of the second object.

20. The computer-readable medium of claim 19, wherein:

the tag portion includes N bits of the tagged self-relative numeric reference that are less significant than bits used for an offset portion; and

the tag portion contains one of at least a first tag value indicating that the first object is contiguous and a second tag value indicating that the second object is non-contiguous, wherein a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^N$.

21. The method of claim 1, wherein:

tag potions of the tagged numeric reference comprise N bits and store at least a first tag value and a second value indicating complementary properties; and

a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^N$.

22. The method of claim 1, wherein the sum consists of the tagged numeric reference and the second tagged machine pointer.

23. The method of claim 5, wherein:

the memory is subdivided into a plurality of pages;

the first object is stored on a first page; and

the second object is stored on a second page, other than the first page.

24. The computer-readable medium of claim 11, wherein:

tag potions of the tagged numeric reference comprise N bits and store at least a first tag value and a second value indicating complementary properties; and

a difference of the first tag value and the second tag value is congruent to $2^{N-1}$ modulo $2^{N}$.

25. The computer-readable medium of claim 11, wherein the sum consists of the tagged numeric reference and the second tagged machine pointer.

26. The computer-readable medium of claim 15, wherein:

the memory is subdivided into a plurality of pages;

the first object is stored on a first page; and

the second object is stored on a second page, other than the first page.